A. 背包问题I

```
#include<bits/stdc++.h>
using namespace std;

int f[1010], v[1010], w[1010];
signed main()
{
        int n, m;
        cin >> n >> m;
        for(int i = 1; i <= n; i ++)
                cin >> v[i] >> w[i];

        for(int i = 1; i <= n; i ++)
        {
                for(int j = m; j >= v[i]; j --)
                {
                        f[j] = max(f[j], f[j - v[i]] + w[i]);
                }
        }
        cout << f[m] << '\n';
        return 0;
}
```

B. 背包问题II

```
#include<bits/stdc++.h>
using namespace std;

int f[1010], v[1010], w[1010];
signed main()
{
        int n, m;
        cin >> n >> m;
        for(int i = 1; i <= n; i ++)
                cin >> v[i] >> w[i];

        for(int i = 1; i <= n; i ++)
                for(int j = m; j >= v[i]; j --)
                        for(int k = 0; k * v[i] <= j; k ++)
                                f[j] = max(f[j], f[j - k * v[i]] + k * w[i]);
        cout << f[m] << '\n';
        return 0;
}
```

C. 背包问题III(Easy)

```cpp
#include<bits/stdc++.h>
using namespace std;

int f[110], v[110], w[110], s[110];

signed main()
{
        int n, m;
        cin >> n >> m;
        for(int i = 1; i <= n; i ++)
                cin >> v[i] >> w[i] >> s[i];

        for(int i = 1; i <= n; i ++)
        {
                for(int j = m; j >= v[i]; j --)
                {
                        for(int k = 0; k * v[i] <= j and k <= s[i]; k ++)
                        {
                                f[j] = max(f[j], f[j - k * v[i]] + k * w[i]);
                        }
                }
        }
        cout << f[m] << '\n';
        return 0;
}
```

D. 背包问题III

```cpp
#include<bits/stdc++.h>
using namespace std;

int f[2010], v[2010], w[2010], s[2010];

int new_v[25000], new_w[25000], cnt;

signed main()
{
        int n, m;
        cin >> n >> m;
        for(int i = 1; i <= n; i ++)
        {
                cin >> v[i] >> w[i] >> s[i];
                int k = 1;
                while(s[i] >= k)
                {
                        cnt++;
                        new_v[cnt] = k * v[i];
                        new_w[cnt] = k * w[i];
                        s[i] -= k;
                        k *= 2;
                }
                if(s[i] > 0)
                {
                        cnt++;
                        new_v[cnt] = s[i] * v[i];
                        new_w[cnt] = s[i] * w[i];
                }
        }


        for(int i = 1; i <= cnt; i ++)
        {
                for(int j = m; j >= new_v[i]; j --)
                {
                        f[j] = max(f[j], f[j - new_v[i]] + new_w[i]);
                }
        }
        cout << f[m] << '\n';
        return 0;
}
```

E. 背包问题IV

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 110;

int n, m;
int v[N][N], w[N][N], s[N];
int f[N];

int main()
{
    cin >> n >> m;

    for (int i = 1; i <= n; i ++ )
    {
        cin >> s[i];
        for (int j = 0; j < s[i]; j ++ )
            cin >> v[i][j] >> w[i][j];
    }

    for (int i = 1; i <= n; i ++ )
        for (int j = m; j >= 0; j -- )
            for (int k = 0; k < s[i]; k ++ )
                if (v[i][k] <= j)
                    f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);

    cout << f[m] << endl;
    return 0;
}
```

F. 最长上升子序列

```
#include<bits/stdc++.h>
using namespace std;

int a[1010], f[1010];

signed main()
{
        int n;
        cin >> n;
        int ans = 0;
        for(int i = 1; i <= n; i ++) cin >> a[i];
        for(int i = 1; i <= n; i ++)
        {
                f[i] = 1;
                for(int j = 1; j <= i - 1; j ++)
                {
                        if(a[j] < a[i])
                                f[i] = max(f[i], f[j] + 1);
                }
                ans = max(f[i], ans);
        }
        cout << ans << '\n';
        return 0;
}
```

G. 最长上升子序列(HardVersion)

```
#include<bits/stdc++.h>
using namespace std;

int a[100010];

signed main()
{
        int n;
        cin >> n;
        vector<int> stk;
        for(int i = 1; i <= n; i ++) cin >> a[i];
        stk.push_back(a[1]);
        for(int i = 2; i <= n; i ++)
        {
                if(a[i] > stk.back()) stk.push_back(a[i]);
                else
                {
                        auto it = lower_bound(stk.begin(), stk.end(), a[i]);
                        *it = a[i];
                }
        }
        cout << stk.size() << '\n';
        return 0;
}
```

H. 数字三角形

对于当前的点 (i,j) 来说,它只能从 $(i-1,j-1)$ 和 $(i-1,j)$ 走过来, 所以我们只需要比较这两个点的最大值, 加上当前点的值, 更新 $dp[i][j]$ 即可。

```
#include <iostream>
using namespace std;

const int N = 510,INF = 1e9;
int m[N][N],dp[N][N];

int main()
{
    int n;
    cin >> n;
    for(int i = 0;i<N;i++)
        for(int j=0;j<N;j++)
            dp[i][j] = -INF;
    for(int i = 1;i<=n;i++)
        for(int j=1;j<=i;j++)
            cin >> m[i][j];
    dp[1][1] = m[1][1];
    for(int i = 2;i<=n;i++)
        for(int j=1;j<=i;j++)
            dp[i][j] = max(dp[i-1][j-1],dp[i-1][j])+m[i][j];
    int res = -INF;
    for(int i=1;i<=n;i++)
        res = max(res,dp[n][i]);
    cout << res;
    return 0;
}
```

I. 最短编辑距离

状态表示: $f[i][j]$ 表示将 s 的前 i 个字符变成 t 的前 j 个字符所需要的最少操作次数。

状态转移:

1. 删除操作：把$a[i]$删掉之后$a[1\sim i]$和$b[1\sim j]$匹配,所以之前要先做到$a[1\sim (i-1)]$和$b[1\sim j]$匹配:

$$f[i][j]=f[i-1][j]+1$$

2. 插入操作：插入之后 $a[i]$ 与 $b[j]$ 完全匹配，所以插入的就是 $b[j]$, 那填之前 $a[1\sim i]$ 和 $b[1\sim (j-1)]$ 匹配:

$$f[i][j]=f[i][j-1]+1$$

3. 替换操作：把 $a[i]$ 改成 $b[j]$ 之后想要 $a[1\sim i]$与 $b[1\sim j]$ 匹配,那么修改这一位之前，$a[1\sim (i-1)]$应该与$b[1\sim (j-1)]$匹配:

$$f[i][j]=f[i-1][j-1]+1$$

但是如果本来 $a[i]$ 与 $b[j]$ 这一位上就相等，那么不用改，即:

$$f[i][j]=f[i-1][j-1]$$

细节问题(初始的状态是什么):

1. f[0][i]如果a初始长度就是0，那么只能用插入操作让它变成b
2. f[i][0]同样地，如果b的长度是0，那么a只能用删除操作让它变成b

```
#include <iostream>
#include <cstdio>
using namespace std;

const int N = 1010;
char a[N], b[N];
int n, m;
int dp[N][N]; // dp[i][j]表示把字符串a的1～i个字符改成字符串b的1～j个字符所需要的最小步骤

int main()
{
        cin >> n;
        cin >> a + 1;
        cin >> m;
        cin >> b + 1;
    // 初始化
    for (int i = 0; i <= n; i++)
        dp[i][0] = i; // 如果要把前a的前i个字符改成没有字符，需要全部删掉
    for (int i = 0; i <= m; i++)
        dp[0][i] = i; // 如果要把没有字符改成b的前i个字符需要每个都增加

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
        {
            dp[i][j] = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1); // 由递推可知，在当前情况下a和b的第1～i-1个字符已经完成匹配，因此只需要考虑
                                                                // 和dp[i][j-1]修改(删除或者替换)一次转移过来的所花费次数较小的一种情况即可


            if (a[i] == b[j]) // 如果a的第i位和b的第j位刚好成功匹配，则说明不需要修改，可以考虑从dp[i-1][j-1]直接转移
            {
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1]);
            }
            else // 否则还需要考虑是否需要增加字符
            {
                dp[i][j] = min(dp[i - 1][j - 1] + 1, dp[i][j]);
            }
        }
    cout << dp[n][m];
    return 0;
}
```

J. 斐波那契数列

```cpp
#include<bits/stdc++.h>
using namespace std;

int f[5000010];

const int mod = 1e9 + 7;

int dfs(int x)
{
        if(f[x] > 0) return f[x];
        f[x] = (dfs(x - 1) + dfs(x - 2)) % mod;
        return f[x];
}

signed main()
{
        cin.tie(0) -> sync_with_stdio(0);
        int T;
        cin >> T;
        f[1] = f[2] = 1;
        while(T--)
        {
                int x;
                cin >> x;
                cout << dfs(x) << '\n';
        }
        return 0;
}
```

K. 滑雪

```cpp
#include <cstring>
#include <iostream>
using namespace std;

const int N = 610;
int h[N][N];
int dp[N][N]; //dp[x][y]表示从x,y出发，能走过的最长距离
int n, m;
int dx[4] = {0, 1, -1, 0}, dy[4] = {1, 0, 0, -1};
int dfs(int x, int y) //求解dp[x][y]
{
    int &v = dp[x][y];
    if (v != -1) //如果dp[x][y]已经被求出来了，则直接返回求得的值
        return v;
    v = 1;
    for (int i = 0; i < 4; i++) //如果还没有被求出，则求解dp[x][y]
    {
        int a = x + dx[i], b = y + dy[i]; //模拟移动
        if (a >= 0 && a < n && b >= 0 && b < m && h[x][y] > h[a][b]) //如果目标位置合法(在nm图内而且低于起始位置)
            v = max(v, dfs(a, b) + 1); //则dp[x][y]能划过的最大值等于目标位置能划过的最大值+1
    }
    return v;
}

int main()
{
    memset(dp, -1, sizeof dp); //初始化dp表为-1,表示每一个位置都没有被求过
    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cin >> h[i][j];
        }
    }
    int res = 1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            res = max(res, dfs(i, j));
        }
    }
    cout << res;
    return 0;
}
```